SANDIA REPORT
SAND98–2264
Unlimited Release
Printed October 1998

Connecting Remote Clusters with ATM

Tan Chang Hu and Peter S. Wyckoff

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND98-2264
Unlimited Release
Printed October 1998

Connecting Remote Clusters with ATM

Tan Chang Hu
Advanced Network Integration

Peter S. Wyckoff
Distributed Computing

Sandia National Laboratories
P.O.Box 5800
Albuquerque, NM 87185-0806

Abstract

Sandia's entry into utilizing clusters of networked workstations is called Computational Plant or CPlant for short. The design of CPlant uses Ethernet to boot the individual nodes, Myrinet to communicate within a node cluster, and ATM to connect between remote clusters. This SAND document covers the work done to enable the use of ATM on the CPlant nodes in the Fall of 1997. The report is divided into four major sections:

1. Software and hardware selection and installation.
2. Device Driver port to the Alpha architecture.
3. Porting the ATM Signaling code to the Alpha architecture.
4. Computing aspects/SC 1997 demonstration.

The work was split between the Sandia Networking groups located in California and New Mexico. Pete Wyckoff (Department 8950) from California did the device driver port and the Linux-ATM kernel patch and T. C. Hu (Department 4616) from New Mexico did the ATM signaling port. The conclusion drawn from this effort is that running ATM with Linux/Alpha is feasible, if sufficient resources are allocated to maintain the ATM code to ensure a production quality environment.

CONTENTS

INTRODUCTION

The high cost of purchasing and maintaining a single, powerful, high-performance architectured computer in addition to the advancement of the microprocessor has pushed the concept of utilizing clusters of networked workstations into the foreground. A number of research projects have emerged in recent years to explore this possibility of low cost, clustered computers. Sandia's entry into this arena is called Computational Plant (CPlant).

The current CPlant configuration consists of 18 node "scaleable units": 16 compute nodes, one I/O node, and one service node. The nodes are Digital Equipment 433 Personal Workstations mounted eight to a rack. Each node has two network interfaces. The Tulip (DEC 21143 class) Ethernet interface is used for network booting and other service activities, while the Myrinet interface is used for application message passing. The installed base operating system was Linux Redhat version 4.2, which uses Linux kernel 2.0.30.

The option of utilizing ATM for the communications between remote clusters spanning New Mexico to California was explored. Myrinet is a proprietary technology and could not traverse the public, long haul, telecommunications services. Sandia National Laboratories has the ATM infrastructure already in place between the remote sites and has been actively utilizing it for several years. The device driver port was performed on the California scalable unit and the ATM signaling port was done in the New Mexico Laboratory setting initially and then installed on the scalable units to support SuperComputing '97 demonstration.

SOFTWARE AND HARDWARE SELECTION AND INSTALLATION

The development hardware platform was built to duplicate the CPlant environment. Two Digital Equipment Miatas and the Redhat 4.2 version for Alpha computers were selected as the baseline. For the ATM interface, the Efficient ENI-155P-MF-S was selected, as there was a device driver already available as part of Werner Almesberger's ATM version 0.31 code for the Intel Platform. A purchase order was placed for the hardware and software. However, after the software arrived, the instructions to install Linux per the Redhat manual did not work as expected. This was due to the relatively new Miata platform, which was release around April 1997. There are two ways of booting the Miata: 1) using the System Reference Manual Console (SRM); or, 2) using the AlphaBIOS.

The initial installation ran into several problems. First, the Redhat CDROM image for booting from the SRM would not work with the platform. Once that hurdle was surmounted via booting using the AlphaBIOS boot method, a problem with the Powerstorm graphic card was encountered. The display mode of the card used black letters on a black screen. Next, the system disk was too large for the "fdisk" program and manual intervention was necessary to partition the disk correctly. Lastly, the 12X speed IDE CDROM drive provided intermittent CDROM access that would hang the whole system during the installation. To circumvent the CDROM inconsistency, the CDROM was accessed through the "ftp protocol" using another Linux system as the ftp server. The Network Interface Card (NIC) installation introduced additional complications. PCI slot selection restrictions and deficiencies of the SRM ROM to permit work-around, had to be corrected prior to the successful installation of the NIC. The installation sequence is described chronologically in the paragraphs below.

SRM Installation failure

Following the Redhat instructions for Alpha computers, a typical SRM installation generated the following failures:

The boot image disk was created on an Intel platform.
The Redhat CDROM was mounted on /cdrom.

```
>mount -t iso9660 /dev/cdrom /cdrom
```

Copy /cdrom/milo/images/miata.img to the boot floppy.

```
>dd if=miata.img of=/dev/fd0
```

Insert the boot floppy into the Miata floppy drive and boot via the SRM prompt:

```
>>>boot dva0
(boot dva0.0.0.0.1 -flags a)
block 0 of dva0.0.0.0.1 is not a valid boot block
>>>
```

The Miata image did not contain a valid boot block.

The MILO boot method was tried next.
The /cdrom/milo/miata/milo.dd image was copied over to a floppy.

```
>dd if=milo.dd of=/dev/fd0
```

Insert the MILO floppy into Miata floppy drive and boot from the SRM prompt:

```
>>>boot dva0
(boot dva0.0.0.0.1 -flags a)
reading 772 blocks from dva0.0.0.0.1
bootstrap code read in
base=14c000, image_start=0, image_bytes=60800
initializing HWRPB at 2000
initializing page table at 13e000
initializing machine state
setting affinity to the primary cpu
jumping to bootstrap code

[....Miata freezes here ....]
```

As an off-the-wall test, booting directly from the CDROM was attempted and failed as expected:

```
>>>boot dka0 -fi /images/miata.img -fl load_ramdisk=1
(boot dka0.0.0.0.1 -file /images/miata.img -flags
load_ramdisk=1)
block 0 of dka0.0.0.0.1 is not a valid boot block
>>>
```

AlphaBIOS Installation

The AlphaBIOS route was selected next since the SRM boot failed. A SRM variable had to be reset first to permit jumping to the AlphaBIOS. The DEC documented instructions recommended the following:

From the SRM boot prompt, the "os_type" environmental variable had to be set to "nt". This toggles the system to boot via the AlphaBIOS instead of from the SRM.

```
>>>set os_type nt
```

Power off, wait 5 seconds to power back on.

After setting the SRM to the correct mode, the following instructions for configuring and running Linux in the AlphaBIOS mode taken out of the "Alpha Miniloader HOWTO" were followed:

Booting Linux/Alpha on an AlphaBIOS-based system

With the introduction of the XLT series, Digital changed the system console interface for its NT systems from ARC to AlphaBIOS. AlphaBIOS is a screen-oriented interface, which should be more familiar to PC users. This change in console interface necessitates a change in setup procedure for those who wish to run Linux/Alpha on AlphaBIOS-based systems.

First, install the latest version of AlphaBIOS on the system. This can be obtained from Digital's "System Software and Driver Updates" Web page, http://www.windows.digital.com/support/sysoft.htp. Download the ZIP file, unzip, and install as follows:

- Copy the files to a FAT-formatted floppy
- Turn on the system and insert the floppy.
- At the opening screen, press <F2> to go into setup mode
- Select "Upgrade AlphaBIOS"
- Follow the directions on the screen

Once the AlphaBIOS is at the latest revision level, start bootstrapping the system as follows:

Create a FAT-formatted floppy with the following files:

- linload.exe (from this directory)
- MILO (the version appropriate to your system)
- Get MILO from gatekeeper.dec.com at /pub/DEC/Linux-Alpha/Miniloaderv2.0/2.0.29+/miata/ or from whatever ftp sites listed. The files are to be copied onto a floppy disk.

Example:

```
copy linload-1.5.exe a:\linload.exe
copy milo-2028 a:\milo
```

- Turn on the system and insert the floppy. At the opening screen, press <F2> to go into setup mode
- Select "Utilities->OS Selection Setup..."
- Press INSERT to add a new operating system selection
- For "Boot Name", enter something like "Linux". Press TAB to get over to the next field.
- Press down-arrow until the selection for "Boot File" is "A:".
- TAB over to the next field
- Enter "\linload.exe". TAB *twice* (i.e. skip the OS Path load device -- it's irrelevant)

- Enter "\" for the OS Path load file
- TAB to skip "OS Option:" field
- Press ENTER to add the selection.

At this point, AlphaBIOS will probably put up a dialog box labelled "Warning: Operating System Selection not valid!", ignore this error (it's only a problem for NT) and press ENTER to continue.

- Press F10 to save the changes made; press ENTER to confirm the changes.
- Press ESC twice to get back to the opening screen.
- Use the up and down arrows to select the boot selection just added, and press ENTER to boot it.
- AlphaBIOS will load linload, which will in turn load MILO.
- When the MILO prompt appears, proceed with the installation process for a normal ARC-based system.

The instructions successfully allowed MILO to be booted on the Miata. The kernel image was copied to a root image floppy formatted with an ext2fs file-system with the following command sequence:

```
>mkfs -t ext2 /dev/fd0
>mount -t ext2 /dev/fd0 /mnt
>cp miata-kernel-2030-test.gz /mnt/vmlinux.gz
(or your particular kernel image)
>umount /mnt
```

After booting the Miata into MILO, the root image floppy was inserted into the floppy drive and the following command was issued to complete the operating system bootstrap:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0
      load_ramdisk=1 prompt_ramdisk=1
```

The system successfully booted and asked for the root disk (i.e. ramdisk). The ramdisk floppy was inserted; however, here the Redhat 4.2 Miata boot images stopped displaying any information after jumping to the kernel code. At first, the system appeared to have crashed, so several different modified images were tried. The Miata ramdisk images were pulled from gatekeeper.dec.com /pub/Digital/Linux-Alpha (miata-kernel-2029-complete.gz). The new images were compressed gzip image and could be modified for the system configuration. This modification was performed on an Intel system with a kernel that supports the "ramdisk" option.

Example:

```
mv ramdisk.img ramdisk.test.gz
gunzip ramdisk.test.gz
dd if=ramdisk.test of=/dev/ram0 (or /dev/ramdisk)
mount -t ext2 /dev/ram0 /mnt
```

Once mounted, the image could be modified (See Boot/root HowTo).

After several image modification attempts and searching the Alpha newsgroup's posting on the Redhat WWW homepage, the Powerstorm Graphic card that came with the Miata turned out to be the problem. The graphics card was displaying black letters on a black screen. Redirecting the output to the serial ports was tried as a solution.

> Set the console on the ramdisk to symbolically point
> to the serial port /dev/cua0 and go to the SRM prompt and
> set the "console" variable (or disconnect the keyboard).
>
> ```
> >>>set console serial
> ```
>
> The serial port also needed a null modem in its path to a vt102 terminal.

Unfortunately, the install procedure redirects the output to the screen regardless of the serial port settings and this work-around failed. A purchase requisition for the Matrox Millennium video card was issued at this time.

Once the Matrox Millennium card was received and installed on the Alpha, the installation proceeded. The file system format procedure was the next problem encountered. The "fdisk" script could not handle the disk information as valid parameters. Finally, after several experimentation with the settings, pseudo settings of heads= 255, Cylinders=255, and Sector=32 were utilized. A recommendation that was found on the Redhat Archive site stated the following:

> One way to compute the numbers is to use the
> following formula:
>
> - Write down the numbers for heads, cylinders, and sectors per track.
> - Multiply them together. Heads x Cyl x SPT = x
> - Divide x by 63, divide this by 128, write down the result.
> - Use this number for cylinders, and 63 and 128 for the other factors.
> - Go into fdisk, type x for expert mode, help to see the commands, use the numbers you've now got.
> - Return from expert mode to the previous menu.
> - Partition away.
>
> SCSI controllers and disks use logical remapping anyway, so the numbers are a convention, not a necessity.

After the disk partition problem was resolved, one swap and one root partition was created.

The CD-ROM drive (which was an IDE 12X speed device) was the last installation problem encountered. The installation would start and then freeze during the package install/copy onto the hard disk. The failure would occur randomly during the installation. To work around this problem, the CDROM access method was changed to utilize ftp, with the CD-ROM mounted on another Linux machine. The ftp transfer did work and Redhat distribution was successfully installed.

The installation procedure recommended the following for booting of the hard drive:

> If the first partition of the first disk drive is dedicated to a small FAT partition
> for booting (as the installation procedure advises to), and then once Linux is
> installed, copy linload.exe and MILO to this partition. Once shut down, the
> Linux menu selection can be modified to load MILO from this partition as
> follows:

- At the opening screen, select <F2> to go into setup mode
- Select "Utilities->OS Selection setup"
- Highlight the entry for Linux, then press F6 to edit it.
- TAB over to the device portion of the "Boot File" line. Use the up and down arrow keys to select the hard-disk partition where linload and MILO reside (typically "Disk 0 Partition 1" or "Disk 1 Partition 1"). Press ENTER to confirm the selection
- If auto-booting of Linux is desired after MILO is loaded, then TAB over to the "OS Options" line and enter the MILO command to boot the system, e.g. "boot sda2:vmlinux.gz"
- Press ENTER to confirm the selection.
- Press F10 to save the changes. Press ENTER to confirm.

Once this procedure is completed, booting and running Linux on an AlphaBIOS based system should be very similar to doing so on an ARC system.

As the intent was to modify the kernel for ATM, a floppy boot was elected instead of booting from the hard disk. This way, should the developmental kernel fail to boot, there will always be a viable kernel on the hard disk.

Network card installation

For the ATM interface, the Efficient ENI-155P-MF-S was selected as there was a device driver already available as part of Werner Almesberger ATM version 0.31 code for the Intel Platform. There are peculiarities to the Alpha PCI bridge chipset, which required the ATM network card to be inserted into the primary PCI slots for the operating system to access the ATM interface. Otherwise the machine will boot fine but runs into problems as soon as the device driver module loads. Unfortunately, the AlphaBIOS boot complained about an unsupported device in the primary PCI slot (no obvious override mechanism is available) and requires the unknown device to be removed prior to booting the MILO loader.

Booting with the ENI card in the primary slot under AlpaBIOS produced the following error:

```
>>>boot dva0 -file vmlinux.gz -flags "root=/dev/sdb2"
...
Illegal PCI device in slot4. Power down and remove PCI device
```

If the ENI card was placed in a secondary slot the system boots, however, the system will hang on any attempt to load the ENI device driver module.

```
>export MODPATH=/lib/modules/2.0.30/net:/lib/modules/2.0.30/atm
/root/insmod.other suni
>insmod eni

  pw:eni_detect at 0xfffffe00000ab798
  pw:real_base before stripping 0xa400000
  pw:command was 0x46
  pw:trying command 0x146
                         ^
```

```
                    The machine hangs here
```

Although the Redhat installation was installed through the AlphaBIOS route, all subsequent booting had to be switched back to the "aboot"/SRM method. A copy of "aboot" (version 0.5) was pulled from ftp.azstarnet.com. The kernel was recompiled first with CONFIG_ALPHA_SRM defined. The aboot-0.5 binary was then recompiled for this configuration. The "aboot"/SRM boot floppy was successfully built on the Miata under Linux with the following:

```
>mke2fs /dev/fd0
>cd /root/aboot-0.5
>./tools/e2writeboot /dev/fd0 bootlx
>mount /dev/fd0 /mnt
>cp arch/alpha/boot/vmlinux.gz /mnt
>umount /mnt
```

The AlphaBIOS was redirected in the advanced CMOS option (F6 under CMOS) to boot Digital UNIX. Once the system initiated to the SRM bootstrap prompt the system was directed to create a "pci_device_override" environmental variable with the value set to 0002111a. The SRM created the environmental variable, but still failed the PCI check during boot.

Further inquiries to Digital Equipment uncovered the fact that SRM version v6.4-214 or later was required for the pci_device_override to function as intended. The latest firmware was obtained from the gatekeeper.dec.com site.

```
ftp://ftp.digital.com/pub/Digital/Alpha/firmware/readmes/interim/d
pw433/
```

```
ftp gatekeeper.dec.com
cd pub/Digital/Alpha/firmware/readmes/interim/dpw433/
bin
prompt off
mget miata_v662_upd.exe miata_v662_upd.sys mkbootfirm.tar
bye
tar xf mkbootfirm.tar
cp miata_v662_upd.exe fwupdate.exe
```

The firmware files from gatekeeper were installed on a bootable floppy. The SRM settings were recorded first, prior to the update as the upgrade resets the SRM variables.

A low-level format floppy was created in drive zero on the Miata (the Miata was booted into Digital UNIX).

```
fddisk -fmt /dev/rfd0a
```

The firmware update was copied on to the floppy disk.

```
./mkbootfirm/mkbootfirm fwupdate.exe | dd of=/dev/rfd0c bs=64k
```

The system was rebooted to SRM and redirected to boot from the update floppy:

```
>>>boot dva0
```

The next commands were issued to update the ROM's after reboot:

```
APU> prompt:
APU> update
...
APU> update SROM
...
APU> exit
```

The update necessitated a powercycle of the Miata and when SRM came back, an edit of the variables was required to match the previous settings. The default SRM sets auto_action to HALT so manual boot commands can be issued. The default setting for auto_action is RESTART.

Once the update was completed, the Miata loaded the ENI module drivers without a hitch.

```
>/root/insmod.other suni (special insmod version otherwise
                   insmod complains about size problem error)
>insmod eni
```

The module message generate the following messages indicating success:

```
pw:eni_detect at 0xfffffe00000ab798
pw:real_base before stripping 0xa000000
pw:command was 0x46
pw:trying command 0x146
eni(itf 0):rev.0,real_base=0xa000000, irq=28,pw:dense
base=0xfffffc860a000000
pw:phy=0xfffffc860a020000,reg=0xfffffc860a040000,ram=0xfffffc860a2
0000
mem=2048KB (00-20-EA-002D-56)
eni(itf 0):ASIC, MMF
```

DEVICE DRIVER PORT

Porting the Linux-ATM drivers to Alpha had never been done before, and required a large amount of work. The major constraints were to ensure that the components were compatible with the 64-bit Alpha architecture, and that the driver was compatible with other kernel patches, which were necessary for the rest of the kernel to work on an Alpha.

Standard Linux-Alpha improvements

First, a stock 2.0.30 Linux kernel was obtained from a standard ftp site, and at the time, the kernel had no support for the Alpha architecture. Patches to allow for this came from Digital's support site at ftp.gatekeeper.com and came in two bunches.

The first bunch changed the configuration files to support Alpha-specific parameters and added support for some of the many PCI chipsets delivered with Alpha machines: Apecs, CIA, LCA. Various little fixes for the discrepancy between unsigned long on i386 and Alpha were also applied in this bunch, and affected many unrelated parts of the kernel. Proper console support was included (TGA instead of VGA support), as well as differences in the real-time clock.

The second bunch added further support for Alpha chipsets: Noritake, Miata, Sable, and AlphaBook1. This support affects many of the low-level kernel functions including interrupt, booting, and trap handling. The name of the specific PCI chipset, which would cause much grief, was Pyxis, and shipped only in the Miata.

A few other patches were applied to provide support for network booting of the kernel. The SRM console (hardware bootstrap kernel) provides for loading a kernel image from the network; interfacing this into the Linux kernel is the province of the bootp patches applied at this step. Jay Estabrook provided the initial support for bootp, which was then modified to provide support for shipping the initial filesystem image across the network with the kernel. This allows for a completely diskless boot of the initial kernel and filesystem in a scaleable way. An alternative approach is to provide the filesystem via NFS, which is currently used in practice due to its simplicity in allowing quicker debugging of the filesystem setup.

There was a major problem in ensuring compatibility between the SRM console and the Linux kernel. The kernel had an idea of where in physical memory it would end up, only the SRM was very inflexible in its use of memory and would destroy areas of the kernel image by placing its runtime stack on top of the image. Since Digital was unwilling to provide the source for the SRM, the bootp functionality of the kernel was manipulated to work around the inflexibility of the SRM, i.e., the Linux kernel was told to use a different bootstrap address.

ATM specific changes

The Linux-ATM project is led by Werner Almesberger and can be found at http://lrcwww.epfl.ch/linux-atm. The project was originally designed to support only the i386 architecture, and this fact was the root of most problems in porting the software to the Alpha.

The latest distribution of Linux-ATM that was compatible with version 2.0 Linux kernels was 0.31. Included in that distribution was a patch file for the kernel. When applied, many rejected patch hunks were created. Most of these rejected patches were in the IPv4 networking section of the kernel and arose due to code reorganizations from the Alpha patches described above. Linux-ATM also included support for application-requested IP, a means by which a user-level code can cause the kernel to automatically initiate an ATM connection to a remote machine. While this project did not use this functionality, the offending patches were changed to fit the working version of the kernel. Another failed patch was a duplicate from the Alpha-specific patches that was intended to provide improved serial console functionality. The Makefiles and configuration files for the Alpha also required updating to support the same functionality as their counterparts for the i386.

Having resolved the errant elements of the Linux-ATM kernel patch, the next step was to attempt to compile a functional kernel. The largest source of work was hunting down uses of "unsigned long" in the code for 32-bit types. This included looking for architecture-specific 32-bit objects, fixing the translation of virtual to physical memory addresses, and adjusting the translation of main physical memory to PCI bus addresses. Warning messages generated by the Gnu C compiler were very helpful in locating certain types of coding bugs.

On the i386 architecture, the size of an "unsigned long" is 32 bits. This is the natural word size for that hardware that includes the PCI bus and all 32-bit PCI interface cards. (Only recently have a few 64-bit PCI devices and busses entered the commodity market.) On 64-bit architectures, including the Alpha, an "unsigned long" is full 64 bits, its natural word size. Hence many sections of the i386 code where an unsigned long is used should actually directly specify an architecture-specific 32-bit object.

The methodology used to discover these incompatibilities was a brute force search for "long" everywhere in the code, then reading through enough of the surrounding context to determine if it really should be a long (64bits) or if it should be a u32 (32-bit unsigned integer). For instance, where an unsigned long is used to hold a pointer to machine memory, it must be 32 bits on i386 and 64 bits on Alpha, so "unsigned long" is the appropriate type for this case. Network addresses require true 32-bit types on both architectures, and were changed to "u32" in the socket and TCP handling layers.

The driver code for the ATM device itself required the most attention. There are multiple ways of addressing the same memory space in the machine. First there is the physical address for a location in memory, represented by a 32-bit word on i386, and by a 64-bit word on Alpha. The size of this quantity is dictated by the size of an addressable quantity built into the architecture of the CPU chip itself. Second there can be multiple (or no) "virtual addresses" corresponding to a physical address. These are managed by translation mechanisms in the hardware and allow processes to address memory in a natural way. A virtual addressing mechanism permits access to an address space larger than the physical memory available, and gives the kernel flexibility in the way it manages memory. Since this is closely tied to the CPU hardware, the virtual address size is the same as that of a physical address.

The last addressing mode is PCI space addresses. Both the Alpha-based Miata and common i386 machines are built around a 32-bit PCI bus. The CPU to host adapter and all the PCI devices "see" host memory through a 32-bit address space, as that is the largest address size that the bus will allow. Common operations on a PCI bus involve the following sequence of transactions: the CPU allocates a buffer in its physical address space and fills it with data to be transferred to a PCI device. This buffer could be the contents of a network packet to be written out the ATM interface. The CPU then translates the physical address of the buffer into a PCI-space address using a well-known mapping that is setup at boot time through negotiation between the CPU and the PCI bus adapter. The CPU writes this PCI-space address and other control words directly into the PCI device, using a segment of the CPU's virtual address space, which is mapped to the PCI device's physical memory. This set of control information causes the PCI device to initiate a DMA transfer on the PCI bus, reading from the host memory and writing to the ATM interface (or other external connection). When the PCI device finishes the transfer, it interrupts the CPU to indicate that it no longer needs the memory, and the CPU reclaims it for future use.

The complication in converting a Linux i386-specific source code to work on the Alpha platform is that all three address spaces are identical on the former architecture. This means that the programmer need make no firm distinction among the three addressing modes. On the Alpha, however, translations between the addresses must be made explicitly. This is accomplished with simple calls like "virt_to_bus()" and "bus_to_phys()", which perform a few additions and bit shifts, but are critical, unlike on the i386 where the functions were the identity. Further, on the Alpha the sizes of virtual and physical addresses are 64 bits, while a PCI-space address has size 32 bits. On the i386 all three were the same. In porting the code, not only did the three different address spaces had to be disambiguated, the proper type words generated for each space had to be ensured; simply using "unsigned long" everywhere was not sufficient.

Allocations of packet buffers and DMA spaces on the Alpha had to be increased to 64-bit boundaries in all locations. This required changes to many checks in the driver code.

The PCI standard allows for the CPU to find the location (in PCI-address space) of a particular device by reading out of PCI configuration registers. The CPU can then translate this into its own address space and access the PCI device as if it were simply an extension of its own virtual memory space. On the i386 architectures this is trivial, but on the Alpha it requires knowledge of the particular PCI chipset mapping strategies. For the Pyxis used by the Miatas, this is through "dense" memory and there is a linear map based on the offset location of this dense memory space. The term "dense" refers to the fact that the access methods are restricted to those which can be used on main physical memory, i.e., reads and writes must be 64-bit aligned, and at least 32 bits long. (One uses "sparse" memory if byte reads and writes are required, and if the corresponding performance penalty is acceptable). The natural read word size for any 32-bit PCI device mapped through dense memory is 32-bit. What may be possibly considered a bug in the Pyxis chipset is that a 32-bit read of dense memory, actually causes a read of 64 bits, half of which is then discarded. This may not seem such a big issue until one notices that performing a read of certain registers on most PCI cards can not be done without side effects. For instance, on the ENI PCI ATM adapter used in this project, the master control/status register is located at offset 4 (in units of 32-bit words), and the statistics register is located at offset 5. A read of the status register through Pyxis dense memory will cause a read of the statistics register, which is then discarded. However, when the card notices that the statistics register has been read, it clears the value, figuring that the host noticed and took the appropriate measures, when actually just the PCI chipset read and discarded the data. This and other cases can lead to disastrous consequences if not planned for. The work around is to read only 64-bit words and store the unneeded half in a buffer somewhere for the relevant section of code to read when it needs it. Not all registers used on the ENI card operated in this fashion, though, and many headaches were saved by that fortune.

The device driver was loaded as a kernel module so that the system did not have to be rebooted continuously during the port.

Examples of a kernel build

A transcript of a kernel tree build follows, one for the original, and one with the new ATM patches (Comments are delineated by the # sign, the machine prompt is flog$).

```
#
# A pristine linux 2.0.30 version
#
```

```
# linux-2.0.30.tgz - stock source code from sunsite.unc.edu.
# axp-diff-2.0.30.gz - from gatekeeper.dec.com
# axp-patches-2.0.30.gz - from gatekeeper.dec.com
# tulip.c - latest tulip driver from http://cesdis.gsfc.nasa.gov/
#                             linux/drivers/tulip.c
#
# Unpack linux-2.0.30.tgz under /usr/src and add patches:
#
flog$ tar xfz linux-2.0.30.tgz
flog$ mv linux stock
flog$ cd stock
#
# Various alpha and bootp patches, applied to both trees.
#
flog$ gzip -dc ../axp-diffs-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../axp-patches-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../bootp-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../bootp-pw-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../bootp2-pw-2.0.30.gz | patch -p1 -s -N -E
flog$ cp /root/tulip.c-079 ./drivers/net/tulip.c
flog$ vi tulip.c (and add change "static constant
rx_copybreak=16384")
#
# Save source tree
#
flog$ cd ..
flog$ tar xfz linux-2.0.30.tgz
flog$ mv linux pw
flog$ cd pw
flog$ gzip -dc ../axp-diffs-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../axp-patches-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../bootp-2.0.30.gz | patch -p1 -s -N -E
flog$ gzip -dc ../bootp-pw-2.0.30.gz | patch -p1 -s -N -Eflog$
gzip -dc ../bootp2-pw-2.0.30.gz | patch -p1 -s -N -E
flog$ cp /root/tulip.c-079 ./drivers/net/tulip.c
flog$ vi tulip.c (and add change "static constant
rx_copybreak=16384")
flog$ cd ..
#
# atm-0.31-stock.gz is the kernel patch file from atm-0.31.tar.gz
# distribution named atm/atm.patch there.  Many patch errors.
#
flog$ cd stock
flog$ gzip -dc ../atm-0.31-stock.gz | patch -p1 -s -N -E
3 out of 5 hunks failed--saving rejects to net/ipv4/af_inet.c.rej
1 out of 3 hunks failed--saving rejects to net/ipv4/udp.c.rej
Ignoring previously applied (or reversed) patch.
2 out of 2 hunks ignored--saving rejects to
drivers/char/console.c.rej
1 out of 11 hunks failed--saving rejects to
net/ipv4/tcp_input.c.rej
1 out of 3 hunks failed--saving rejects to
net/ipv4/tcp_output.c.rej
flog$ cd ..
```

```
#
#
# pw-hacked patch, applies cleanly (and works).
#
flog$ cd pw
flog$ gzip -dc ../atm-orig-0.31-and-pw-2.0.30.gz | patch -p1 -s -N
-E
flog$ cd ..
#
# Did make config; make dep; make clean; make boot.
#
flog$ exit
```

Some of the files changed were the following:

Check files for differences (verify whether it is OK to add):

arch/alpha/boot/tools
object.c - minor header addition.

arch/alpha/kernel/
bios32.c - code section depends on MIATA definition.
kgdb.c - minor print statement changed.
pyxis.c - addition of debug code DBG_MCK check.
setup.c - additional arch check, change of root device from sda to hdc plus bootp code (initrd stuff).
traps.c - additional printk statement.

arch/alpha/mm/
init.c - initrd stuff (bootp) again.

drivers/atm/
Config.in - minor addition to PCI, add MIATA as alternative.
atmtcp.c - added comments.
eni.c - extended modification.
eni.h -extended modification.
midway.h - extended modification.
suni.c - extended modification.

drivers/block/
rd.c - ramdisk and bootp changes. Not really needed unless using initrd.

drivers/net/
tulip.c - replace with current version.

drivers/scsi/
scsi.c - pointer recast change.

fs/
binfmt_em86.c - removal of a char pointer.

fs/nfs/
nfsroot.c - NFS defines.

nclude/asm-alpha/
atomic.h - minor definition added to shut up gcc.
resource.h - KERNEL limit changes.
unistd.h - libc added, changes to inline int execve.

include/linux/
atmdev.h - changes to int type (__u32 etc) and protoypes.
skbuff.h - unsigned long to __u32 changes.
socket.h - ifdef KERNEL addition. Commented out Ipv6 using ifdef.

init/
main.c - initrd (bootp).

lib/
inflate.c - address inflate problem for initrd. Added gunzip error messages.

net/atm/
atmarp.c - addition of segment.h.
atmarp.h - added comment.
common.c - added segment.h, definition for DPRINTK, alpha specific changes.
common.h - addition of segment.h.
ipcommon.c - added 2 header file changes, and two printk statements.
ipcommon.h - removed linux/atmclip.h.
lec.c - removed segment.h header file.
proc.c - added segment.h header file, redefined proc_atm_read prototype/ function.
pvc.c - addition of segment.h.
raw.c - redefinition of atm_peek_aal0, atm_peek_aal5, and a DPRINTK statement.
svc.c - addition of segment.h, alpha specific get_fs_long to get_fs_quad conversions.

net/core/
skbuff.c - assignment change to shut up gcc.

net/ipv4/
af_inet.c - CONFIG_AREQUIPA stuff.Not really needed needed unless using it.
ip_fragment.c - assignment added during variable declaration.
packet.c - assignment added during variable declaration.
tcp_input.c - add back tcp_options(sk,th); statement. rejected as part of Werner's ATM patch.
udp.c - change sk->ip_route_cache = rt; to set_rt_cache(sk,rt);

net/unix/
af_unix.c - variable declaration moved.

The only significant one is in the net/ipv4/tcp_output.c for CONFIG_SCALED_WINDOWS. Had to add a missing #endif definition.

PORTING THE ATM SIGNALING CODE

As discussed before, the Linux-ATM project was originally designed to support only the i386 architecture. Again, the use of 32bit versus 64bits accounted for the major problems encountered in porting the signaling software to the Alpha. For simplicity and performance sake, only the signaling and classical IP portions of Almesberger's code were ported over to the Alpha platform. The porting of the signaling code was started once Pete Wyckoff's work with the kernel and device driver were installed on a stand-alone Miata . The three daemons, atmsigd, ilmid, and atmarpd contained the ATM code of interest. Atmsigd handled the lower ATM signaling (SAAL, UNI, etc.), ilmid handled the ILMI exchange between the host and the switch, and atmarpd handled the classical IP over ATM services. The daemons had to be started in sequnce and all three have to be operating for Classical IP over ATM to work.

The daemons were reviewed and a number of calls were changed to use size_t and ssize_t as argument types to convert 32bit representation in the code to 64bit to make the code more 64bit safe. Once the code successfully compiled, the signaling services were turned on with logging enabled for debugging purposes. In several instances, a Network General in-line ATM sniffer had to be attached to the Alpha to study the signaling and IP packet flow. These two tools were invaluable in providing pointers to problems in the code.

The ilmid code was the first fail after successful compilation. The daemon failed to run, and died with a return value of EINVAL. By examining the gnu debugger trace, the problem was traced back to the kernel atm_ioctl function call in /usr/src/linux/net/atm/common.c. Basically the variable "eff_arg" was pointing to an incorrect size and failing the verification check. The problem was corrected by changing the assignment of eff_arg from get_long to get_quad (eff_arg = get_fs_quad(&((struct atmif_sioc *) arg)->arg).

Atmsigd provided the next major bug. The daemon would start, generate a large number of kernel traps and killed itself when the next daemon, atmarpd, is initiated. The code for atmsigd was examined with the debugger and the problem was traced to the poll_loop function in io.c that in turn use recv_kern and recv_signaling function calls. Variable size usages in recv_signaling were tracked down and when necessary converted from "int" to "ssize_t" type. This corrected the atmsigd problem.

The masking/arp table was the last major bug encountered. The ATM host would connect up with classical IP to other host with any IP host portion address of greater than 127, but would not connect to host with the IP host portion less than 128. By examining entries in the ATM arp entry table and tracing the function calls used, the bug was found in the subfunction atmarp_setentry in usr/src/linux/net/atm/atmarp.c The argument variable "ip" was redefined to a 32bit rather than 64bit integer. This corrected the ATM arp table entries and fixed the connection problems.

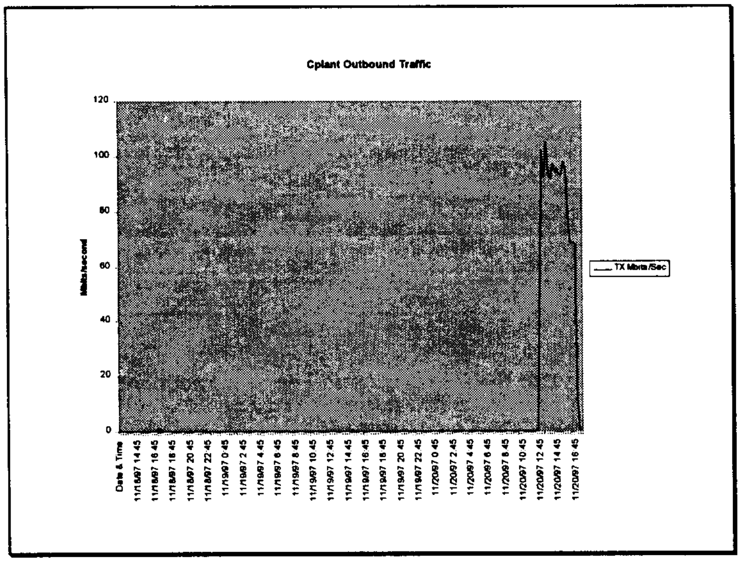
At this point the signaling code was functioning reasonably and throughput numbers were about ~112Mbits/sec for transmit and ~100 Mbits/sec for receives. However, kernel unaligned messages were generated constantly. The problem was traced using the debugger to the SSCOP_PAD and SSCOP_N macros in pdu.c under the atm/saal directory. The culprit appears to be the ntohl() library calls. A side effect of the Alpha version of ntohl() calls is to carry the signed bit. The Intel version did not. This generated an incorrect value during the pad variable calculation (all ones turned the pad to be equal to 255 whenever a one is in the highest bit order), which in turn screwed up the alignment. According to the email exchange between Almesberger and the netdev group for gcc, the ntohl() function had been redefined as of glibc version 2.0.3 to use a 32bit argument. The glibc version provided with Redhat 4.2 was 1.X and did not have the correct definition. Unfortunately, attempts to upgrade the libc on the Miata broke the operating system. As the code does function, the upgrade the libc will be delayed until CPlant moves to the next release from Redhat.

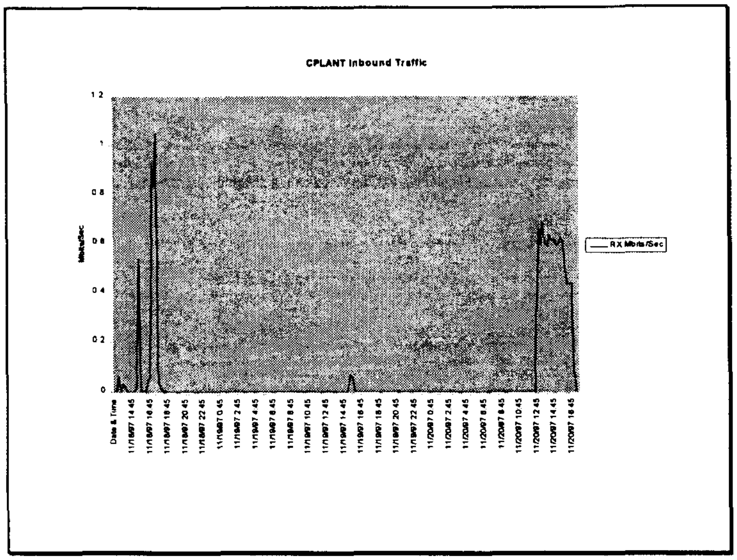
COMPUTING ASPECTS/SC97 DEMONSTRATION

Jay Estabrook sent Sandia National Laboratories a new version of the SRM on Oct 14, 1997. This allowed both the Myrinet and ATM card to be installed in the primary PCI slots instead of just one device at a time. Prior to the installation of the new SRM version 6.6-89 on the Alpha, only the Myrinet or the ATM card was permitted in the primary slots since only one override was permitted. With the new SRM and setting the pci_device_override on the SRM to -1, both cards could be utilized simultaneously.

The boot image and the ATM signaling code were transferred onto two clusters once the ATM code was up and running on the test systems. One cluster was located in California and the other in New Mexico. Part of the California cluster was relocated onto the SC'97 show floor in San Jose, CA. The two clusters were connected with PVC's at various line speeds [1].

The data collected during SC'97 is shown in figure 1. The application was the Smooth particle hydrodynamics (SPH) code written by Pete Wyckoff. The traffic data collected was for the showfloor inbound and outbound. The figure shows asymmetry as the application was primarily pushing data out and receiving acknowledgments back. For the time period that the application was actively running, the sustained output averaged approximately 93Mb/s.





[figure 1]

One interesting side note was that the ATM code on the Linux box behaved better under stress with the FORE ATM switch than with a Cisco switch. A trace with the network general showed that the Linux to switch connection gets dropped when bombarded with 64k ICMP packets. When the Linux system tries to re-establish the connection via SSCOP, the Cisco switch ignores the SSCOP messages. For future reference, this will have to be resolved to enable the use of current generation Cisco switches.

CONCLUSION

Utilizing ATM with Linux as a transport medium is a viable option based on the results obtained from the SC'97 demonstration. However, both the Linux kernel and the ATM code are changing and evolving constantly. As of November 19, 1997, the ATM code was advanced to version 0.33 with the development Linux kernel 2.1.65 in mind. To utilize ATM in any type of production environment, continuing technical support for porting the Linux kernel, ATM NIC device driver, ATM signaling code, and resolving incompatibilities between the ATM switch and ATM code are necessities. This may change in the future as Werner Almesberger is working on including the ATM code as part of the standard Linux kernel distribution. The expectation is that the throughput will continue to increase and the stability to improve as the ATM code matures.

REFERENCES

[1] Pete Wyckoff and Rob Amstrong, *Smooth particle hydrodynamics on ATM-connected distributed PC clusters*, submitted to 1998 International Conference on Parallel and Distributed Processing Techniques.

Distribution

1 - 0469 R. J. Detry, 2900
1 - 0469 J. F. Jones, Jr., 4600
1 - 0318 G. S. Davidson, 9215
1 - 0321 A. L. Hale, 9224
1 - 0321 W. J. Camp, 9204
1 - 0630 P. J. Vandevender, 4010
1 - 0741 S. G. Varnado, 6200
1 - 0801 M. J. Murphy, 4900
1 - 0806 C. D. Brown, 4621
1 - 0806 J. P. Brenkosh, 4616
1 - 0806 J. M. Eldridge, 4616
10 - 0806 S. A. Gossage, 4616
1 - 0806 T. C. Hu, 4616
1 - 0806 L. G. Martinez, 4616
1 - 0806 J. H. Naegle, 4616
1 - 0806 L. G. Pierson, 4616
1 - 0806 L. Stans, 4616
1 - 0806 T. D. Tarman, 4616
10 - 0806 M. O. Vahle, 4616
1 - 0806 T. J. Pratt, 4616
1 - 0807 I. C. Alexander, 4617
1 - 0807 R. M. Cahoon, 4418
1 - 0812 L. F. Tolendino, 4914
1 - 0812 M. R. Sjulin, 4914
1 - 0826 J. D. Zepper, 9136
1 - 0836 J. H. Biffle, 9121
1 - 0841 P. J. Hommert, 9100
2 - 0899 Technical Library, 4916
1 - 1111 S. S. Dosanjh, 9221
1 - 1212 J. M. McGlaun, 5903
1 - 1221 J. S. Rottler, 5400
1 - 9003 D. L. Crawford, 5200
1 - 9011 H. Y. Chen, 8910
1 - 9011 J. A. Hutchins, 8930
1 - 9011 J. Costa, 8920
1 - 9011 J. M. Brandt, 8910
1 - 9011 P. W. Dean, 8910
1 - 9011 R. D. Gay, 8920
1 - 9011 R. E. Palmer, 8901
1 - 9011 P. R. Bryson, 8910
1 - 9011 P. S. Wyckoff, 8980
1 - 9018 Central Technical Files, 8940-2
1 - 0619 Review & Approval Desk, 12690
For DOE/OSTI, (2)